Two-Dimensional Mesh Embedding for B-spline Methods

KARIM SHARIFF[1] & ROBERT D. MOSER[2]

[1] *NASA-Ames Research Center, Moffett Field, CA 94035*

[2] *Dept. of Theoretical and Applied Mechanics, Univ. Illinois at Urbana-Champaign, 104 South Wright Street Urbana, IL 61801-2935*

Many factors motivate consideration of B-splines as basis functions for solving partial differential equations. These are arbitrary order of accuracy and high resolving power similar to that of compact schemes. Furthermore, if one uses a Galerkin scheme one gets, in addition to conservation of the discretized quantities, conservation of quadratic invariants such as energy. This work develops another property, namely, the ability to treat semi-structured embedded or zonal meshes for two-dimensional geometries. This can drastically reduce the number of grid points in many applications. An algorithm is presented for constructing a global spline basis that automatically has $d-1$ continuous derivatives at mesh-block boundaries as everywhere else (here d is the polynomial degree). The basis functions are simply suitable products of one-dimensional B-splines. Both integer and non-integer refinement ratios are allowed across mesh blocks. Finally, test cases for linear scalar equations such as the Poisson and advection equation are presented.

1. Introduction

When gradients become large in a certain direction, structured meshes allow one to cluster grid lines. This is inefficient if the regions of high gradient are local in the other directions. For instance in a turbulent boundary layer, streamwise gradients are large close to the wall but small away from the wall. Considerable savings can be obtained by using the semi-structured approach of embedded meshes for a domain divided into blocks or zones within each of which the mesh is regular.

In a previous paper, Kravchenko *et al.* [1] developed a technique for achieving "one-dimensional" mesh embedding for B-splines. "One-dimensional" refers to the fact that

mesh zones were slices which spanned the domain in the other two directions. In particular, Fourier expansions were used in the directions parallel to the wall and B-splines were used in the wall-normal direction. The present work develops a technique for two-dimensional embedding with B-splines. Specifically, an algorithm is provided for generating a spline basis where the domain is partitioned into arbitrary rectangles. The basis automatically has the same high degree of continuity (C^{d-1}, where d is the polynomial degree) at zonal boundaries as everywhere else. Refinement ratios between mesh blocks are not restricted to being integer. The basis may then be used to form differential operators using a Galerkin or collocation scheme.

Let us briefly sketch where the B-spline technique lies in relation to other schemes.

(i) There is a strong similarity in the 1D periodic case between Padé or compact schemes (Lele [2]) and schemes resulting from B-splines. Swartz and Wendroff [3] have shown that both have higher resolving power than an explicit central difference scheme of the same order. We expect that resolving power will be good even in the non-periodic case and with embedding. It should be noted that for the same matrix bandwidth, Padé schemes have a little better resolving power than Galerkin B-spline schemes (see Table I in ref. [3]). (ii) The desired global order of accuracy is arbitrary (an input parameter). (iii) For non-periodic problems, compact schemes require formulation of separate boundary schemes which are not required for B-splines. However, compact schemes employ only a series of 1D matrix inversions and are therefore cheaper. (iv) When a Galerkin scheme is applied to a conservation equation, the corresponding quantities (such as mass and momentum) are conserved in any sub-region of the domain where unity is exactly representable (which is the case for splines). By "conserved" we mean that the rate of change of the total reduces to consistent boundary fluxes. For finite-difference or finite volume methods "simultaneous achievement of both conservation [across mesh blocks] and accuracy is very difficult and even impossible in most cases" according to Kallinderis [5]. In many such methods the order of accuracy drops at mesh interfaces. Such schemes often update each zone separately and interpolate zone boundary information in a separate step. This requires down-wind differencing at some interfaces which can be destabilizing. Such procedures are not needed here but the price to pay is matrix inversions. A further advantage of the Galerkin scheme

is conservation of quadratic invariants (such as kinetic energy for incompressible flow in the inviscid limit). This protects the scheme against non-linear instability ("aliasing"). It also makes the scheme more robust for it produces a sure indicator of lack of resolution, namely, energy accumulation at small scales. For finite-differences, even for regular meshes, conservation of energy has been achieved only for the second-order staggered "pressure" scheme. (vi) In a finite element method, the solution within each element is represented in terms of nodal values located within or on the boundary of the element. This representation is constructed in such a way that at an element edge the solution depends only on the nodal values along the edge. Thus C^0 continuity across elements is obtained along the entire edge. With a B-spline basis, while nodal values are never explicitly employed in the representation, the dependence of the representation on nodal values is global. C^{d-1} continuity is obtained across elements (by "element" we mean the region of support of each piece of the piecewise polynomial). The resolving power of B-splines is a direct result of this higher continuity. In Hermite finite elements, continuity of higher derivatives is obtained at the cost of having nodal derivatives as additional degrees of freedom. For instance in 1D, using nodal values and derivatives as degrees of freedom produces C^1 cubics. By contrast, B-splines give a more refined space, namely C^2 cubics with the number of degrees of freedom still equal to the number of intervals (plus 3). Matrix structure and bandwidth for a Galerkin formulation are also quite different. Cubic C^1 (Hermite) finite elements have between four and six non-zero entries per row, while cubic B-splines produce a heptadiagonal matrix with about half as many rows. What is gained by finite elements is geometric flexibility. In the present method, only a single mapping of the domain from cartesian coordinates is permitted. (vii) The *h-p* finite element method (e.g. Devloo *et al.* [4]) is specifically designed to treat embedded meshes. The modifier "*h*" refers to local spatial refinement: each side of an element can have more than one element on the other side thus leading to "hanging nodes", i.e. nodes which aren't shared by all their neighboring elements. To maintain C^0 continuity across elements in this arrangement requires that the solution value at the hanging nodes be constrained. The modifier "*p*" refers to the fact that an element may neighbor an element having different polynomial order. Again C^0 continuity across elements requires constraints on some nodal values. The present method

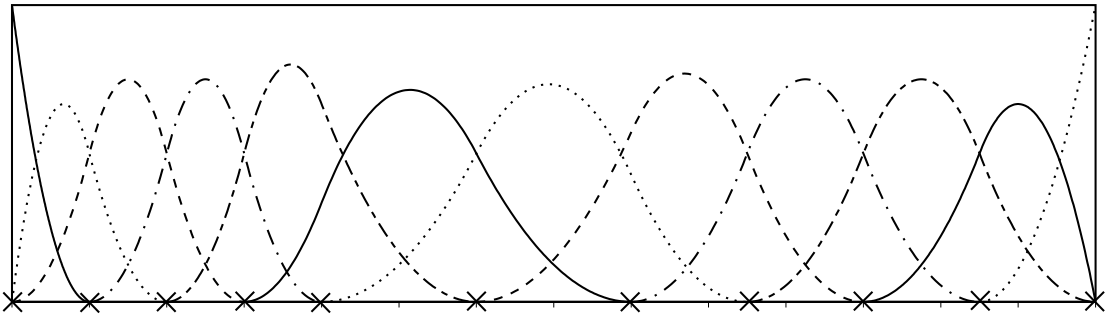

FIGURE 1. Plot of one-dimensional quadratic B-splines on the set of knot points indicated by $\times$. All functions, except those near the boundary, have support over three intervals. Note: Line types are re-used for different functions.

achieves only the h-functionality but with higher continuity.

In summary, it may be said that the B-spline method lies somewhere between finite element and spectral methods in both resolving power and geometric and gridding flexbility.

The outline of this paper is as follows. Section 2 provides a very brief background on one-dimensional B-splines. Section 3 presents an algorithm for choosing an appropriate set of two-dimensional functions and then presents test cases for linear scalar operators.

2. Background on one-dimensional B-splines

For the purposes of this work, a one-dimensional spline is defined to be a polynomial of degree d in each interval with $d-1$ continuous derivatives across interval boundaries. The boundaries of the intervals are called knot-points: the dth derivative of the spline has a jump at the knot points in the interior of the domain.

A *B-spline* is simply defined as a spline which has support over a minimum number of intervals and which is normalized. By equating the number of known continuity and normalization conditions to the number of unknown coefficients one finds that the number of minimum intervals is $d+1$. This is enough information to determine the B-spline which has support in the set of intervals. Quadratic B-splines have support over three intervals and Figure 1 shows the set of quadratic B-splines for the knot points indicated by $\times$. Near the boundary, the number of continuity conditions available drops and so does the number of intervals of support. To calculate the B-splines ones does not have to actually solve any continuity conditions. Rather, one uses a recurrence relation given in de Boor [6] (Chapter 10) to build up the functions from piecewise constant functions.

Above, the B-splines were merely defined as splines with minimum support. What makes them useful for solving partial differential equations is the result, due to Curry and Schoenberg (see de Boor [6]), that they form a *basis* for spline functions with the given knots.

3. Function selection algorithm for 2D mesh embedding

3.1. Mesh Definition

It is assumed that the computational domain is mapped to a rectangle in ξ, η. The case of a more general polygon with right angles (such as a backward-facing step) can perhaps be treated along very similar lines but this is not presently allowed. For convenience, and without loss of generality, the user is required to specify the mesh in terms of sets of points which are swept across certain intervals in order to produce mesh lines. For instance in Figure 2, the mesh lines are indicated by the solid lines. The horizontal lines of the mesh can be generated by sweeping the three sets of η points indicated by $\bullet$ across the ξ intervals indicated. Similarly, the vertical lines can be generated by vertically sweeping the two sets of ξ points indicated by $\times$. Just as in the 1D case where the interval boundaries are knot points, we want mesh lines in the 2D case to represent knot-lines, i.e., the lines normal to which the selected functions (of degree d, say) have a jump in the dth normal derivative. Hence we regard each set of points used to sweep out the mesh as being a knot-set with an associated set of one-dimensional functions. The first ξ knot-set in Figure 2 is the same as the knot-set shown in Figure 1 and has the same associated functions.

If every ξ knot-set either contains or is contained by every other ξ knot-set (and similarly for η), we will say that the mesh is hierarchical. Figure 2 shows a non-hierarchical mesh and the same algorithm applies for both types of meshes.

It is assumed that the mesh thus specified by sweeping points produces closed cells. Suppose that a vertically swept point ξ_1 "hangs" at η_1, i.e. fails to continue into the next sweep. Then to ensure a closed cell, η_1 should be a knot point in the η knot-set which sweeps over ξ_1 or begins or ends a sweep there.

As a preliminary, the algorithm breaks up the mesh into a set of blocks on each of which the mesh is regular. For instance, Figure 2 has five blocks. This decomposition

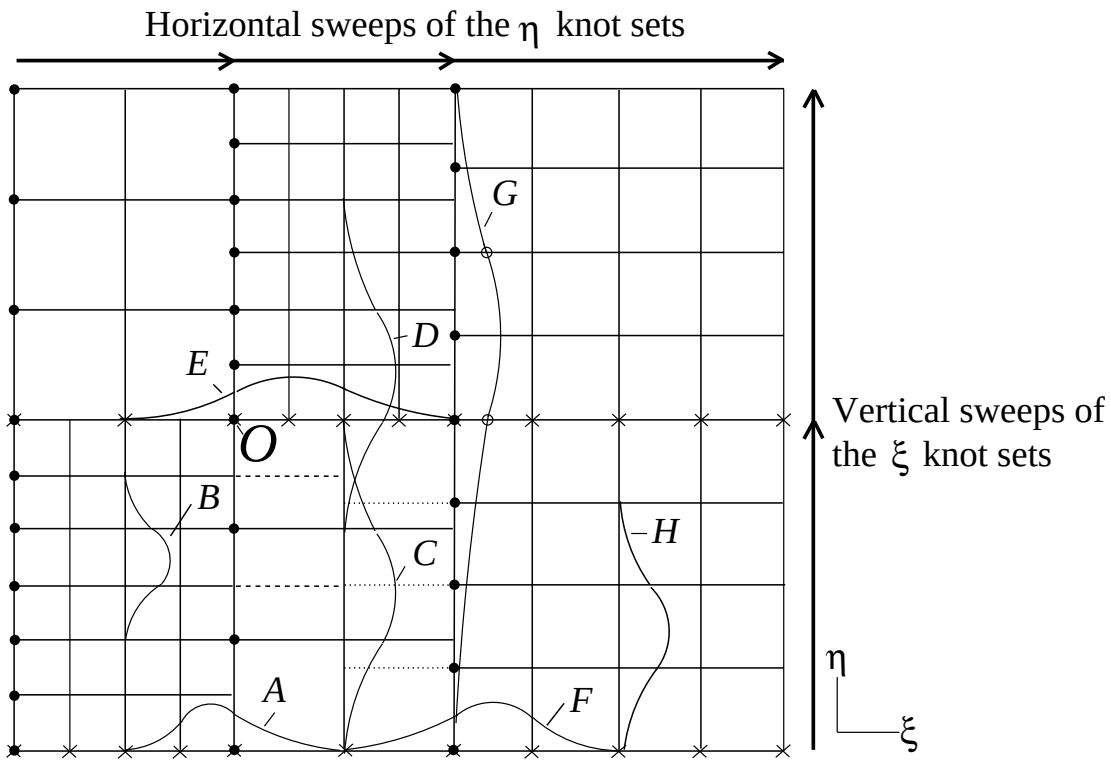

FIGURE 2. Sketch to illustrate the function selection algorithm. All functions shown are quadratic B-splines and span three knot intervals.

is not always unique and should be performed to minimize block boundaries. In the present implementation, a provisional set of blocks is first created on the basis of the sweep intervals. Where possible, blocks are then merged with neighbors to create larger blocks.

Below, an algorithm with variants is presented for selecting functions. The first one, the intersection procedure, does not allow functions that create extra knot-lines not defined by the mesh. For non-integer refinement ratios, this restriction would produce coarse functions along block boundaries and therefore less constrained procedures are described in §3.3.

3.2 Intersection procedure

We want to choose a set of functions $B_n(\xi, \eta), n = 1, 2, \ldots N$ that can represent piecewise polynomials of degree d having knot lines that coincide with the mesh. We have no formal proof that the procedure provides a complete basis. Each $B_n(\xi, \eta)$ is constructed as a product of one-dimensional B-splines, $f(\xi)g(\eta)$, say. One may consider implementing a

brute-force algorithm in which all possible pairs of ξ and η functions defined by the swept knot-sets are tried and those that create undesired knot lines are rejected. This procedure would not only be too costly but does not allow the choice of functions not defined by the given knot-sets which, we shall see, come into play for non-hierarchical meshes.

Two-dimensional functions confined to each block are chosen *a priori*: these are simply the tensor product B-splines for a regular mesh. It remains to select functions that penetrate multiple blocks. They are referred to as "spilling functions". The procedure is to consider every function, say $f(\xi)$, on each sweep of the ξ knot-sets and find suitable functions of η as multipliers. The same procedure is repeated for every η function on each sweep of the η knot-sets. Constant reference is made to Figure 2.

(1) First, one obtains the η knot-set from which multipliers for $f(\xi)$ will be chosen. The ξ-support of $f(\xi)$ will penetrate a certain range of sweep intervals of the η knot-sets. For instance the function A penetrates sweeps 1 and 2 of the η knot-sets. To prevent creation of new knot-lines, we must choose multipliers from the *intersection* of the η knot-sets associated with the range of sweeps. We will call this the *compatible knot-set* of $f(\xi)$. For instance in Figure 2, the function B (from the first η knot-set) is not a compatible multiplier for A because the resulting product has additional knot-lines indicated by the dashes. However, the function C (from the intersection of η knot-sets 1 and 2) *is* a compatible multiplier. Note that if the knot-sets involved in the intersection operation are hierarchical (i.e. each set either contains or is contained in another) then the intersection operation just chooses the coarsest of the sets. In general, however, the compatible knot-set could turn out to be one that was not used in the mesh definition.

    At each zonal boundary there is a strip within the region of fine resolution into which the coarse functions penetrate. The width of this strip is d intervals normal to the boundary. For instance, A is the left-most function in ξ knot-set number 1 that will multiply coarse functions in η. Thus functions (such as $A \times C$) that have the y knot spacing of the coarse block penetrate two ξ intervals of the fine block. In general the number of intervals of penetration is the polynomial degree, d.

(2) Only those functions in the compatible knot-set that have support in the sweep interval

of the knot-set of $f(\xi)$ are relevant; we refer to these as *compatible functions.*

(3) Next, we further limit the compatible functions in order to prevent block-confined functions, which have been chosen *a priori*, from being selected. The support of $f(\xi)$ $\times$ its η sweep interval is either confined to a block or it is not.

   (a) If it is confined to a block then only those compatible functions that cross the block boundaries in η are allowed.

   (b) If it is not confined to a block (as is the case for function A), then all compatible functions are allowed.

(4) In order to prevent the selection of 2D functions with additional knot-lines, it is necessary that the compatibility be *mutual.* For instance D is a compatible multiplier of A, but A is not a compatible multiplier of D. This is because function D penetrates sweeps 1 and 2 of the ξ knot-sets but the knots of function A do not belong to their intersection. In particular, the following test is applied: do the knots of $f(\xi)$ belong to the compatible knot-set of $g(\eta)$?

   The mutual compatibility test needs to be relaxed at non-hierarchical corners. Consider, for instance, the region near O where two fine regions meet at a corner. The only functions that have support at the corner O that do not result in additional knot-lines are functions such as $E \times D$, neither of which belongs to any knot-set used in the definition of the mesh. Since only functions on the knot-sets used in the mesh definition seek suitable multipliers, such a product would never be chosen. This is overcome by either of two modifications, denoted as M1 and M2. If the mutual compatibility test fails for a prospective 2D function containing a block corner *and* the $g(\eta)$ does not belong to any of the η knot-sets penetrated by the support of $f(\xi)$ then: (M1) $g(\eta)$ is chosen, resulting in the creation of new knot-lines, or, (M2) The product of $g(\eta)$ and all the functions on the compatible knot-set of $g(\eta)$ that have support at the corner are chosen. In this case no new knot-lines are created in the context of the intersection procedure. The unmodified procedure is denoted as M0.

(5) Finally, a check is made that a 2D function selected is unique.

3.3. Less constrained procedures

It may be desirable to change resolution gradually, in a non-integer fashion, as is the case in Figure 2 between the left and right halves of the mesh. If compatible multipliers for function F were chosen according to the intersection procedure, the very coarse function G would result (its knots are indicated by $\circ$). To avoid this, one can dispense with the requirement that no new knot-lines be created. Instead of using the intersection operation one can use any other operation which provides sufficiently good resolution along the block boundary. One simple way of doing this is to use the "densest" instead of the intersection operation to determine the compatible knot-set. In other words choose the set with the most knots. A more logical choice is to assemble the compatible knot-set by taking, within each sweep, the set which has the most points within that sweep. For example, the compatible knot-set for function F would be formed by taking points from the rightmost η knot-set for the first vertical sweep interval and the middle η knot-set within the second vertical sweep interval. We refer to this as the "densest by sweep (DS)" operation. In this case a function such as H would be a valid multiplier for F and their product would create the additional knot-lines indicated by dots. The function H is also mutually compatible with F because the knots of F belong to the (single) ξ knot-set penetrated by H. The mutual compatibility test was needed in the intersection algorithm to prevent additional knot-lines. Here it is necessary to prevent selection of multiple types of products in the same region. For instance, function C would seek multipliers at some point. One sees that F would not be a mutually compatible function because the knots of C do not belong to the DS of the vertical knot-sets on which F has support. This is rightly so because $F \times H$ is the type of product we have selected for this region.

In the tests that follow the algorithm used will be denoted by a prefix: I (intersection), D (densest), or DS (densest on a sweep by sweep basis). This will be followed by a suffix (M0, M1 or M2) to denote the modification to the mutual compatibility test.

Algorithm summary: Perhaps the following summary will aid the reader in holding the algorithm firmly in mind. For every ξ function on the knot-sets whose ξ support $\times$ η sweep interval is confined to a block, pick as multipliers only those mutually compatible η functions which cross the block boundaries in η since the rest produce 2D functions

confined to the block and have been chosen *a priori*. For a ξ function whose ξ support $\times$ η sweep interval is not confined to a block, choose as multipliers *all* mutually compatible functions. Repeat the procedure for all η functions defined by the given knot-sets. If required, relax the test of mutual compatibility according to M1 or M2.

3.4. Matrix structure of linear operators

In a Galerkin (weighted residual) method, linear operators (such as the Laplacian and advection operator in the present examples) give rise to matrices of the following general form:

$$(\mathcal{L}_1 B_m, \mathcal{L}_2 B_n) \equiv \int \mathcal{L}_1 B_m \, \mathcal{L}_2 B_n \, dxdy, \quad m, n = 1, 2 \ldots N, \tag{1}$$

where $\mathcal{L}_1$ and $\mathcal{L}_2$ represent linear differential operators. Similarly, operators with a quadratic non-linearity give rise to integrals of triple products. All matrices need be computed once. Since differentiation does not alter the support region of a function, all linear operators produce matrices with the same structure that depends on which pairs of functions overlap. The template for this structure is constructed once. Pairs of functions confined to each block overlap in a simple way and produce the structured part of the matrix: it has $(d+1)^2$ diagonals for a symmetric operator and $(2d+1)^2$ diagonals for a non-symmetric operator. Spilling and block-confined functions that overlap produce scattered matrix elements as do two overlapping spilling functions. With the template in hand, matrix elements are computed using Gauss quadrature with enough points to ensure exact integrals.

The collocation approach can also be used; the peak location of each function is a convenient choice for the collocation points. The matrix for fitting a spline to data given at the collocation points has $(d+1)^2$ bands for the structured part of the matrix.

Matrix equations are solved using the conjugate gradient routines from the SLAP library, which requires that the left-hand-side matrix multiply a vector in each iteration. All the tests reported here used the matrix diagonal as the preconditioner. Incomplete Cholesky decomposition was also tried as a preconditioner. It required a factor of 2 to 3 fewer iterations but the overall CPU time for matrix inversion was a little higher due to the cost of applying the preconditioner at each iteration (even if we leave out the cost of the initial

factorization).

It is important to make the matrix by vector multiplication efficient since it is applied in each iteration. The multiplication of each diagonal can be performed in a vector loop. The template for storing the scattered elements ensures that their multiplication can be accomplished in a certain number of vector multiplies This number is the maximum number of scattered elements in a row. The template is rearranged to avoid the most obvious bank conflict, namely that of accessing the same element of the multiplying or resultant vector within a certain number of CPU clocks.

4. Test cases

First consider scalar advection at 45° to the x-axis:

$$u_{,t} + cu_{,x'} = u_{,t} + c\,(u_{,x} + u_{,y}) = 0 \text{ on the unit square,} \tag{2}$$

where x' is a coordinate at 45° to the x-axis. The initial profile is a Gaussian pulse:

$$u(x, y, t = 0) = e^{-x'^2/\sigma^2}, \tag{3}$$

with $\sigma = .15$. The advection speed, c, is set to unity. The boundary condition imposed on $u_{,t}$ at the in-flow (left and bottom) boundaries corresponds to uniform propagation of a Gaussian pulse. Henceforth, we use as weight functions those B-splines, say $B_m(x, y), m = 1, 2, \ldots N_o$, which vanish where boundary conditions are applied and represent the solution as

$$u(x, y, t) = \sum_{n=1}^{N_o} a_n(t) B_n(x, y) + \sum_{n=N_o+1}^{N} a_n(t) B_n(x, y), \tag{4}$$

where B-splines having non-zero value at the boundary have been isolated in the second term. Denoting the inner product as $(.,.)$, the weak formulation reads:

$$\sum_{n=1}^{N_o} (B_m, B_n)\dot{a}_n = -c \sum_{n=1}^{N} (B_m, B_{n,x} + B_{n,y}) a_n - \sum_{n=N_o+1}^{N} (B_m, B_n)\dot{a}_n, \quad m = 1, 2, \ldots N_o \tag{5}$$

where the coefficients, $\dot{a}_n$, in the last term are known from projecting the boundary specification of $u_{,t}$. In order to make time integration errors smaller than spatial discretization errors, this equation is advanced with a sixth-order Runge-Kutta scheme with

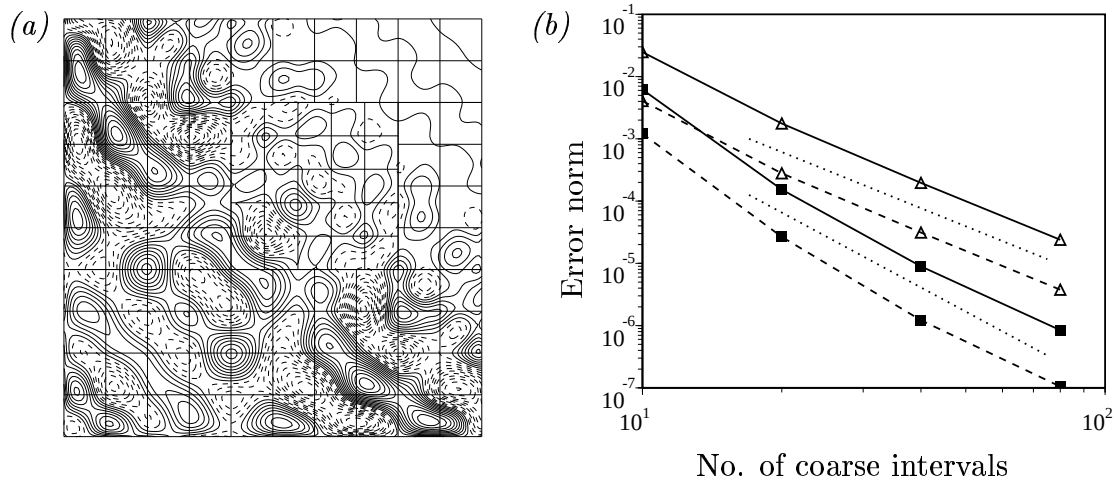

FIGURE 3. *(a)* Local error for the advection equation after the initial pulse has propagated a distance of 0.6. Contour min., max. and inc. $= (-.0065, .0060, .0005)$. The error is sampled on a 150×150 grid. *(b)* Convergence test for the advection equation. Same instant, mesh type and sample points as *(a)*. ——— , Maximum error (L_∞ norm); - - - - , Average of absolute error (L_1 norm); ····· , reference lines with slope of -3 and -4; $\triangle$, for quadratic splines; ▪ , for cubic splines.

cfl $\equiv c\Delta t/\Delta x_{\min} = 0.5$. Figure 3*a* shows contours of local error obtained using cubic splines and algorithm DM0 for a non-integer refinement ratio of 5/4. At the instant shown, the pulse has propagated 4 half-widths and the peak of the pulse lies on the diagonal of the fine block.

The error is perfectly symmetric about the 45° line, smooth, without any peculiarities at the mesh interface, and attains a maximum of 0.65% even on the rather coarse mesh. Figure 3*b* is the result of a convergence test using quadratic and cubic splines. Aside from a little curvature and flattening for the cubic case, the approximation-theoretic convergence rate, $d+1$ (e.g. see Strang[8] , p. 62) is obtained.

Next consider the Poisson equation for the streamfunction given the vorticity, ω:

$$\nabla^2\psi(x,y) = -\omega(x,y) \text{ on the unit square } \Omega \text{ with } \psi = g(s) \text{ on } \partial\Omega. \tag{6}$$

In order to eliminate unknown boundary terms in the weak formulation and to impose the Dirichlet boundary condition strongly we again use as weight functions all the B-splines which vanish at the boundary. The weak formulation reads:

$$\sum_{n=1}^{N_o} c_n(\nabla B_m, \nabla B_n) = (B_m, \omega) - \sum_{n=N_o+1}^{N} c_n(\nabla B_m, \nabla B_n), \quad m = 1, 2, \ldots N_o \tag{7}$$

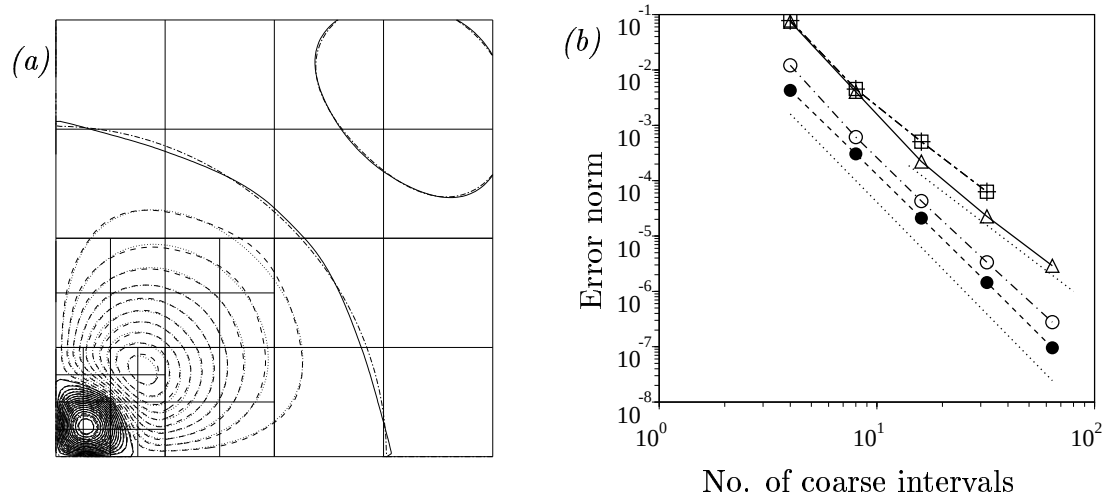

FIGURE 4. Three-vortex Poisson test. *(a)* Solution sampled on a 100×100 grid. The actual mesh is shown. Computed solution: ---- , negative contours; ——— , positive contours. Exact solution: ····· , negative contours; —·— , positive contours. *(b)* Convergence. Error sampled at 512×512 points: —·— □ , Maximum error for the embedded mesh; —·— + , Maximum error for a uniform mesh having the finest spacing of the embedded mesh. Error sampled at all the "mesh-points": ——— △ , maximum error (L_∞ norm); —·— ○ , r.m.s. error (L_2 norm); ---- • , Average of absolute error (L_1 norm). ····· , reference lines with slopes of -3 and -4.

The test vorticity field in the present example consists of three axisymmetric Gaussians of alternating sign and graded intensity superposed with images to make the left and bottom boundaries impermeable walls:

$$\omega(x,y) = \sum_{p=1}^{3} \sum_{i=-1}^{1} \sum_{i=-1}^{1} \frac{ij\Gamma_p}{\pi\sigma_p^2} \exp\left(-r_{pij}^2/\sigma_p^2\right), \qquad r_{pij}^2 = (x - ix_p)^2 + (y - jy_p)^2. \tag{8}$$

The outer sum in (7) is over the three vortices and the two inner sums are over the four images of each vortex. The strengths of the vortices were chosen to be $\Gamma_1 = 1, \Gamma_2 = -9$ and $\Gamma_3 = 12$. The locations and core sizes were set to $x_1 = y_1 = \sigma_1 = 0.5$, $x_1 = y_1 = \sigma_1 = 0.125$, and $x_3 = y_3 = \sigma_3 = 0.0625$. The exact streamfunction resulting from each vortex is easily obtained and the sum is

$$\psi(x,y) = \sum_{p=1}^{3} \sum_{i=-1}^{1} \sum_{i=-1}^{1} -\frac{ij\Gamma}{4\pi} \left[\log r_{pij}^2 - Ei(-r_{pij}^2/\sigma_p^2)\right] \tag{9}$$

where Ei is the exponential integral. In the numerical solution the condition $\psi = 0$ is imposed on the left and bottom walls and the exact solution is imposed on the top and right boundaries.

Figure 4*a* shows that even for the very coarse mesh, the agreement between the exact and numerical solutions is excellent (the intersection algorithm (IM0) with quadratic B-splines was used). For comparison, the solution was also obtained for a uniform mesh having everywhere the mesh spacing of the finest block of the embedded mesh. The maximum error occurs in the intense vortex near the corner and it has virtually the same value for the two meshes (compare $\square$ and $+$ in Figure 4*b*; the error was sampled on a 512×512 set of points for both meshes).

The rest of the curves in Figure 4*b* show that when the error is evaluated at the "mesh-points" (i.e. the intersection points of the knot-lines), two of the norms converge at fourth order which is one order higher than the approximation theoretic result. This phenomenon is called super-convergence and deserves a brief comment. Thomée [9] showed that in one-dimension and with periodic boundary conditions, the Galerkin B-spline method exhibits super-convergence at the knot points. In particular for the heat equation the convergence rate is $2d$ while for the advection equation it is $2d + 2$. Exact time integration is assumed in both cases. Various numerical tests we performed indicated that super-convergence was practically non-existent for two dimensions with non-periodic boundary conditions: Figure 4*b* represents, indeed, an exceptional case. For the advection equation there was no super-convergence even on a uniform mesh. For the Poisson equation on a uniform mesh, the odd-degree splines considered did not exhibit superconvergence, while among the even-degree splines only the quadratic functions displayed a rate consistent with Thomée's result. Quartics converged at sixth order compared with fifth order for approximation theory and the eighth order of Thomée's result.

Figure 5 shows the computational cost for the Poisson test on a CRAY C90 single processor. The cases are the same as those shown in the convergence plot (Figure 4*b*). In a time-dependent problem, the set-up cost of the matrix, which is an order of magnitude larger than the cost of matrix inversion, would be amortized over the number of steps. For the left-most data point, 80% of the cost of computing the Laplacian matrix comes from the unstructured elements even though they constitute only 7% of the matrix. This is because their 1D integrals involve products of B-splines from different knot sets and these integrals are computed as they arise. For the structured elements, however, each

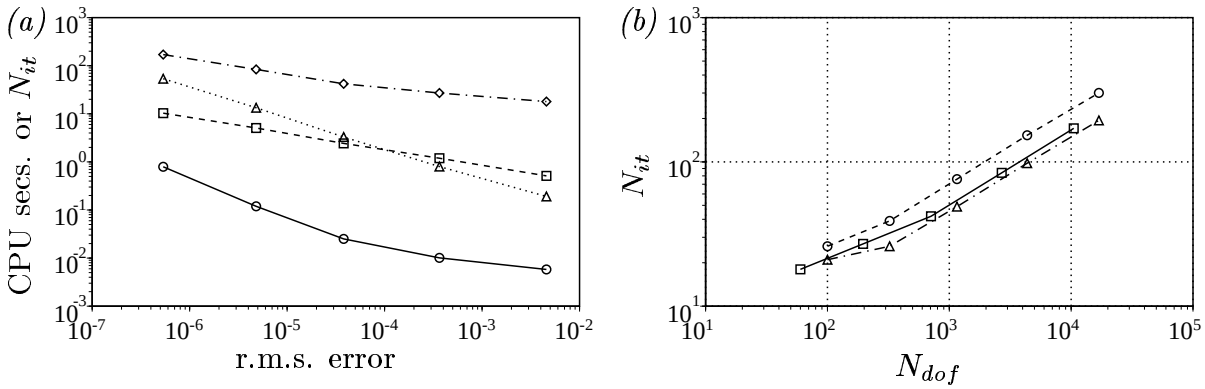

FIGURE 5. *(a)* Computational cost for the Poisson test. ——— ○ , CPU secs. for matrix inversion; - - - - □ , CPU secs. for set-up of Laplacian matrix; ····· △ , CPU secs. for set-up of right-hand-side vector; —·— ◇ , Number of conjugate gradient iterations (N_{it}). *(b)* Number of conjugate gradient iterations for different mesh types. —·— △ , uniform mesh; ——— □ , embedded mesh; - - - - ○ , stretched non-embedded mesh.

1D integral involves B-splines from the same knot set, and B-spline values at quadrature points as well as 1D integrals can be pre-computed for each knot set.

In response to a referee, we determined whether there is a degradation in the convergence of the conjugate gradient method for an embedded mesh. Three types of meshes were compared: the embedded mesh of Figure 4, a regular but non-uniform mesh obtained by eliminating the embedding (resulting in the mesh shown in Figure 6*b*) and finally, a uniform mesh. Each mesh was successively refined and the number of iterations were plotted versus the number of degrees of freedom (N_{dof}). The result is shown in Figure 5*b*. The number of iterations rises as $AN_{dof}^{1/2}$ and A is largest for the non-uniform regular mesh, smallest for the uniform mesh, with the embedded mesh lying between the two. CPU time was also investigated and a mechanism for degradation on a vector machine was uncovered for small meshes. As N_{dof} was decreased, the cost of multiplying each non-zero matrix element increases for each mesh due to shortening vector-loop lengths. The embedded mesh has the shortest loops and for a rather small mesh with $N_{dof} = 200$, there is a factor of two degradation over the uniform mesh.

In the advection test the local error had no peculiarities near the mesh interface. The same is not true for the Poisson test. Figure 6*a* shows that the error reaches a maximum in the region of the intense vortex nearest the corner and diminishes away from it, as one would expect. However, it increases again in the form of positive and negative layers along

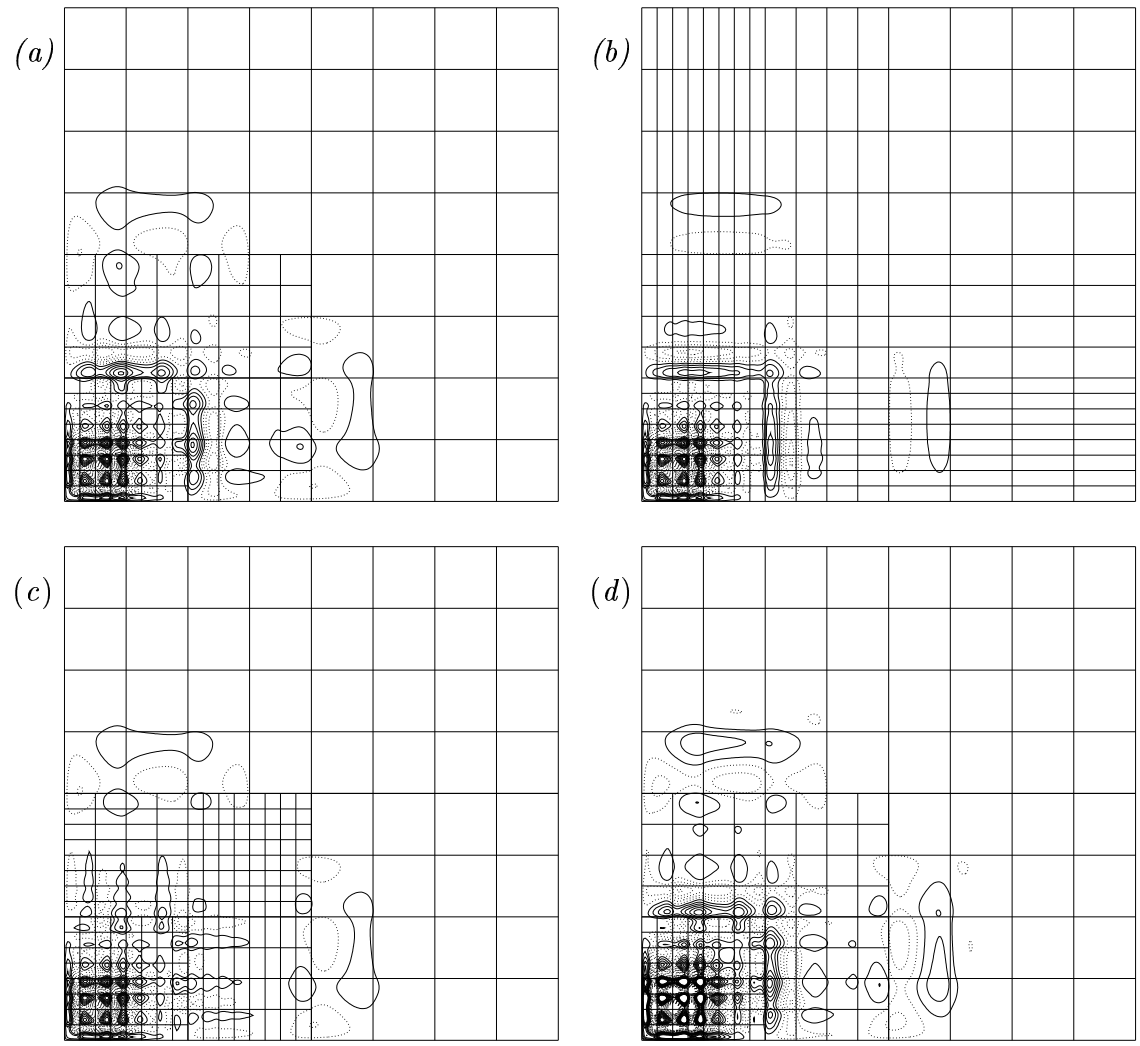

FIGURE 6. Local error on three different meshes for the three-vortex Poisson equation test. In all plots, contour min., max., inc. $= (-.00475, .00425, .00050)$. This makes the lowest contour levels $\pm.00025$ rather than zero. ——— , positive values; ····· , negative values. The error is evaluated on a 300×300 grid. *(a)* Mesh embedding with abrupt changes in spacing normal and tangential to the mesh interfaces. *(b)* A non-embedded but non-uniform mesh with abrupt changes in normal resolution only. *(c)* Mesh embedding with an abrupt change of spacing in the tangential direction in the region of interest. *(d)* The result of adding the error fields shown in *(b)* and *(c)*.

the interface. A regular mesh (without embedding; see Figure 6*b*) but with the same change in spacing *normal* to the interface as the embedded mesh gives similar features in the error but the peak in the error at the mesh interface is a little smaller (70% of the value in the embedded mesh). In Figure 6*c*, normal spacing is kept uniform while *tangential* resolution changes. The pattern of the error near the interface resembles swords pointing normal to

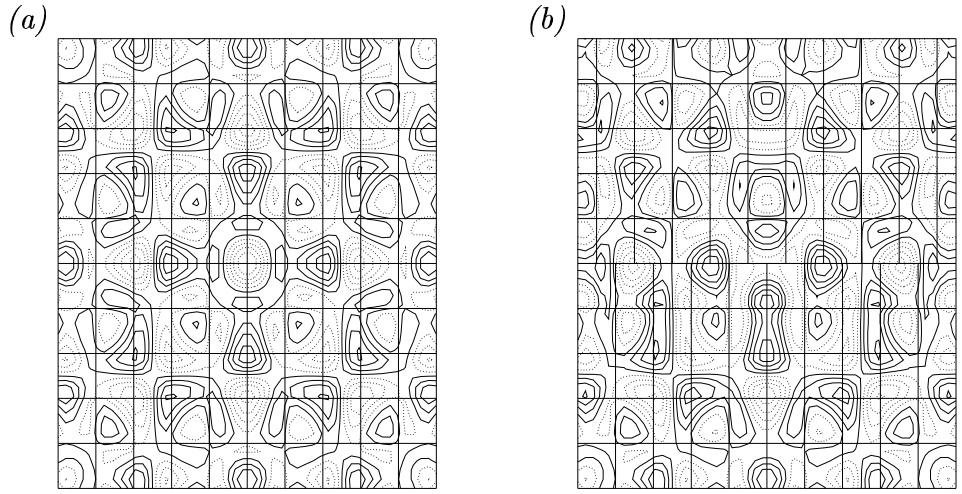

FIGURE 7. Local error for the eigenfunction for $m = n = 6$. ——— , positive values; ····· , negative values. In both plots contour min., max., inc. $= (-.19, .17, .04)$ which makes the lowest contour levels $-.03$ (dashed) and .01 (solid) rather than zero. For plotting purposes, the error was evaluated on a 50×50 grid. *(a)* Uniform mesh *(b)* Dislocated mesh.

the interface with the positive peak near the interface being half the value in the original embedded mesh. Figure 6*d* shows that near the interface of interest, the two errors very nearly add to give the error in the original embedded mesh.

The final test is one of robustness: we consider a problem for which a uniform mesh is optimal and study how much the resolving power degrades when the mesh is dislocated (as shown in Figure 7*b*) and the embedding procedure is applied. A uniform mesh is optimal for problems in which the solution oscillates uniformly everywhere in the domain. The eigenvalue problem for the Laplacian operator is one such problem:

$$\nabla^2\psi = \lambda\psi \text{ on a rectangle of unit width and height } h, \text{ with } \frac{\partial\psi}{\partial n} = 0 \text{ on the boundary.} \tag{10}$$

The von Neumann boundary condition was chosen because it makes the boundary term of the weak formulation vanish. The square, h^2, of the height of the domain is chosen to be irrational ($h^2 = \sqrt{2}$) to avoid degenerate eigenvalues. The exact eigensolutions are:

$$\lambda_{m,n} = -\pi^2(m^2 + (n/h)^2), \quad \psi_{m,n} = \cos(m\pi x)\cos(n\pi y), \tag{11}$$

The extent to which a numerical method is able reproduce the exact eigenvalues and eigenfunctions is a test of its resolving power for the Laplace operator (with the given boundary conditions) across all scales. We compare performance on a uniform 10×10

Mesh type	10% error tolerance	1% error tolerance	No. of degrees of freedom
Uniform	$n_\lambda = 90 \quad n_v = 63$	$n_\lambda = 30 \quad n_v = 22$	144
Dislocated	$n_\lambda = 93 \quad n_v = 57$	$n_\lambda = 29 \quad n_v = 22$	151

TABLE 1. Number of eigenvalues (n_λ) and eigenfunctions (n_v) with error less than the specified tolerances. Relative error is used for eigenvalues; L_2 error is used for eigenfunctions.

interval mesh (without embedding) with performance on the dislocated mesh shown in Figure 7*b*.

To compute the error for each member in the set of numerical eigensolutions one needs to associate it with an exact eigensolution. Any procedure one uses to pair a numerical eigensolution with an exact one necessarily becomes arbitrary for the highly inaccurate eigensolutions. The procedure we used was to evaluate each numerical eigenfunction on a 50×50 grid, normalize it to have a value of unity at the origin, and pair it with the closest (in the discrete L-2 sense) unpaired exact eigensolution. The pairing proceeded in order of increasing numerical eigenvalue. To assess resolving power, the number of eigenvalues and eigenvectors which satisfy a given error tolerance for the two meshes are shown in Table 1. For a tolerance of .01, the number of "good" eigensolutions is almost identical for the two meshes. The overhead of extra functions at the dislocation increases the number of degrees of freedom from 144 to 151. Therefore the ratio of good eigenvectors to the total number of degrees of freedom degrades by 5% for a tolerance of .01. For a tolerance of .10 this ratio degrades by 14%. Figure 7 plots the error in an eigenfunction whose L_2 error degrades by 3% with embedding. This eigenfunction has three wavelengths over the domain in each direction. Aside from the broken symmetry, the error along the dislocation is very similar in the two meshes.

5. Concluding remarks

A technique has been developed for achieving two-dimensional mesh embedding with B-splines as basis functions. The results of test cases are encouraging and work is under way to apply the technique to the incompressible Navier-Stokes equations for three-dimensional flow in two-dimensional curvilinear coordinates.

Since the cost of selecting functions and computing various matrices is incurred every time the mesh changes, the present method would not be efficient for applications requiring frequent adaptive remeshing. The application we have in mind, namely statistically stationary turbulence, should not require frequent re-meshing. For continuous adaptation, an alternate approach to constructing the basis that allows greater flexibility in "editing" the degrees of freedom and matrices should be pursued. One can look to spline wavelets (e.g. Chui & Wang [9]) but the usually imposed requirement of orthogonality results in wavelets of wide support. Szeliski & Shum [10] (p. 1203) state that they have constructed a non-orthogonal form of spline wavelet with smaller support but no explicit formulas are provided. Forsey & Bartels [11], in the context of surface modeling, exploit the fact that a coarse B-spline can be re-written as a sum of finer B-splines and use this for local refinement. Gornowicz [12], in the context of motion analysis of images (where a certain functional has to be minimized) writes the solution as a summation over resolution levels, keeping all B-splines at each resolution level. The maximum resolution level is varied in different regions of space. As it stands, this approach retains redundant information unlike the wavelet representation where each wavelet space contains only the additional information of a refinement. Using such ideas, an efficient and continuously adapting flow solver could be developed.

References

1. A.G. Kravchenko, P. Moin & R.D. Moser, Zonal embedded grids for numerical simulation of wall-bounded turbulent flows, *J. Comp. Phys.* **127**, 412 (1996)
2. S.K. Lele, Compact finite-difference schemes with spectral-like resolution, *J. Comp. Phys.* **103**, 16 (1992)
3. B. Swartz & B. Wendroff, The comparative efficiency of certain finite element and finite difference methods for a hyperbolic problem, In *Proc. Conference on the Numerical Solution of Differential Equations*, edited by A. Dold and B. Eckmann, Lect. Notes in Math. (Springer-Verlag 1974), vol. 363, p. 153.
4. P. Devloo, J.T. Oden & P. Pattani, An *h-p* adaptive finite element method for the

numerical simulation of compressible flow, *Comp. Meths. in Appl. Mech. & Engr.* **70**, 203 (1988)

5. Y. Kallinderis, Numerical treatment of grid interfaces for viscous flows, *J. Comp. Phys.* **98**, 129 (1992).
6. C. De Boor, *A Practical Guide to Splines* (Springer-Verlag, 1978).
7. Strang, G. & Fix, G.J., *An Analysis of the Finite Element Method* (Prentice-Hall, 1973).
8. V. Thomée, Spline-Galerkin methods for initial-value problems with constant coefficients, in *Proc. Conference on the Numerical Solution of Differential Equations*, edited by A. Dold and B. Eckmann, Lect. Notes in Math. (Springer-Verlag 1974), vol. 363, p. 164.
9. C.K. Chui & J.-Z. Wang, On compactly supported spline wavelets and a duality principle. *Trans. Amer. Math. Soc.* **330**, 903 (1992)
10. R. Szeliski, & H.-Y. Shum, Motion estimation with quadtree splines, *IEEE Trans. Pattern Anal. Machine Intell.* **18**, 1199 (1996)
11. D.R. Forsey & R.H. Bartels, Hierarchical B-spline refinement. In Proc. of SIGGRAPH88. *Comput. Graph.* **22** (4), p. 205 (1988)
12. G.G. Gornowicz, Continuous-field image-correlation velocimetry and its application to unsteady flow over an airfoil. M.S. Thesis, Graduate Aeronautical Labs., Caltech., Pasadena, CA.

Figure captions

FIGURE 1. Plot of one-dimensional quadratic B-splines on the set of knot points indicated by $\times$. All functions, except those near the boundary, have support over three intervals. Note: Line types are re-used for different functions.

FIGURE 2. Sketch to illustrate the function selection algorithm. All functions are quadratic B-splines and span three knot intervals.

FIGURE 3. *(a)* Local error for the advection equation after the wave has propagated a distance of 0.6. Contour min., max. and inc. $= (-.0065, .0060, .0005)$. The error is sampled on a 150×150 grid. *(b)* Convergence test for the advection equation. Same instant, mesh type and sample points as *(a)*. ——— , Maximum error (L_∞ norm); - - - - , Average of absolute error (L_1 norm); ····· , reference lines with slope of -3 and -4; $\triangle$, for quadratic splines; $\blacksquare$, for cubic splines.

FIGURE 4. *(a)* Solution to the three-vortex Poisson equation sampled on a 100×100 grid. The actual mesh is shown. Computed solution: - - - - , negative contours; ——— , positive contours. Exact solution: ····· , negative contours; —— , positive contours. *(b)* Convergence for the three-vortex Poisson equation test. Error sampled at 512×512 points: —-— $\square$, Maximum error for the embedded mesh; —— + , Maximum error for a uniform mesh having the finest spacing of the embedded mesh. Error sampled at all the "mesh-points": ——— $\triangle$, maximum error (L_∞ norm); —·— $\circ$, r.m.s. error (L_2 norm); - - - - $\bullet$, Average of absolute error (L_1 norm). ····· , reference lines with slopes of -3 and -4.

FIGURE 5. *(a)* Computational cost for the Poisson test. ——— $\circ$, CPU secs. for matrix inversion; - - - - $\square$, CPU secs. for set-up of Laplacian matrix; ····· $\triangle$, CPU secs. for set-up of right-hand-side vector; —·— $\diamond$, Number of conjugate gradient iterations (N_{it}). *(b)* Number of conjugate gradient iterations for different mesh types. —·— $\triangle$, uniform mesh; ——— $\square$, embedded mesh; - - - - $\circ$, stretched non-embedded mesh.

FIGURE 6. Local error on three different meshes for the three-vortex Poisson equation test. In all plots, contour min., max., inc. $= (-.00475, .00425, .00050)$. This makes the lowest contour levels $\pm.00025$ rather than zero. ——— , positive values; ····· , negative values. The error is evaluated on a 300×300 grid. *(a)* Mesh embedding with abrupt changes in spacing normal and tangential to the mesh interfaces. *(b)* A non-embedded but non-uniform mesh with abrupt changes in normal resolution only. *(c)* Mesh embedding with an abrupt change of spacing in the tangential direction in the region of interest. *(d)* The result of adding the error fields shown in *(b)* and *(c)*.

FIGURE 7. Local error for the eigenfunction for $m = n = 6$. ——— , positive values; ····· , negative values. In both plots contour min., max., inc. $= (-.19, .17, .04)$ which makes the lowest contour levels $-.03$ (dashed) and .01 (solid) rather than zero. For plotting purposes, the error was evaluated on a 50×50 grid. *(a)* Uniform mesh *(b)* Dislocated mesh.